Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Balažic

Zajem in obdelava podatkov s spleta

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: Prof dr. Marko Bajec

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Splet je skorajda neomejen vir podatkov. Čeprav so za iskanje na voljo odlični spletni iskalniki, nam na koncu še vedno ostane, da vse zadetke, ki jih prek iskalnikov dobimo, preberemo sami in iz njih izluščimo relevantne podatke. V okviru diplomske naloge si zamislite računalniško rešitev, ki bo omogočala avtomatski zajem izbranih podatkov s spletnih strani in njihov prenos v nadzorno ploščo. V nadzorni plošči naj bo moč določati pravila za obveščanje, ki bodo uporabnikom omogočala, da definirajo pogoje (vrednosti posameznih podatkov, ki se objavljajo na različnih spletnih straneh), ob katerih bi želeli biti posebej obveščeni (t.i. alarmi). Izdelajte načrt rešitve in le-to implementirajte v poljubnem razvojnem okolju.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Balažic sem avtor diplomskega dela z naslovom:

Zajem in obdelava podatkov s spleta

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom Prof. dr. Markota Bajeca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 30. avgust 2015 Podpis avtorja:

Želel bi se zahvaliti celi družini za podporo skozi leta mojega študija, še posebej sestri Ani za dobro voljo. Zahvalil bi se tudi mentorju Marku Bajecu za vse nasvete glede diplomske naloge ter Patriciji za vztrajno dopolnjevanje mojih stavkov z vejicami.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Osnovni pojmi in tehnologije **3**

2.1 Semantični splet 3

2.2 Spletno luščenje podatkov 4

2.3 Pajek 5

2.4 DOM 6

2.5 Tehnologije 7

2.6 Razvojna orodja 18

3 Pregled obstoječih rešitev **21**

3.1 Razširitev Scrapy 21

3.2 Safari web clip widget 21

3.3 Firebug 22

3.4 Mediatoolkit 22

3.5 Primerjava 22

4 Idejna zasnova in načrt rešitve **25**

4.1 Uporabniške zgodbe 25

4.2 Shema sistema 26

4.3 Konceptualni model podatkovne baze 27

5 Implementacija **31**

5.1 Domača stran 31

5.2 Preverjanje novih obvestil 33

5.3 Podstran odrezki 37

5.4 Moji odrezki 37

5.5 Prijavni zaslon 40

5.6 Google Chrome razširitev 42

6 Kritičen pogled in možne izboljšave **45**

7 Zaključek **47**

Literatura **49**

Seznam uporabljenih kratic

HTML (angl. HyperText Markup Language) - Standardni označevalni jezik, uporabljen za izdelavo spletnih strani.

XML (angl. Extensible Markup Language) - Razširljiv označevalni jezik definira pravila kodiranja dokumnetov, ki so razumljiva računalniku in ljudem.

XHTML (angl. Extensible HyperText Markup Language) - Ima enak pomen kot HTML, le da je standardiziran s sintakso XML.

CSS (angl. Cascading Style Sheets) - Je slogovni jezik, ki skrbi za stil in prezentacijo spletne strani.

DOM (angl. Document Object Model) - Neodvisni in večdomenski vmesnik za predstavitev objektov in dokumentov, kot so HTML, XHTML in XML.

W3C (angl. World Wide Web Consortium) - Mednarodni inštitut, ki skrbi za razvoj in standarde spleta.

ORM (angl. Object-relational mapping) - Metoda izdelave in poizvedovanja nad podakovno bazo s pomočjo objektono usmerjenega programiranja.

HTTP (angl. HyperText Transfer Protocol) - Glavni protokol za prenos informacij na spletu.

AJAX (angl. asynchrounous Javascript and XML) - Skupina medsebojno povezanih spletnih tehnik, ki se uporabljajo za pošiljanje asinhornih klicev med med strežnikom in odjemalcem.

MVC (angl. Model-view-controller) - Je arhitekturni vzorec za implementacijo grafilnih vmesnikov.

WSGI (angl. Web Server Gateway Interface) - Je preprost in univerzalen vmesnik med spletnimi serverji in spletnimi aplikacijami.

VCS (angl. Version control systems) - Je sistem za upravljanje z različicami dokumentov.

ZIP - Format dokumentov, ki podpira brezizgubno stiskanje.

JSON - (angl. JavaScript Object Notation) - Standardni format za shranjevanje podatkov v človeku berljivi obliki.

URL - (angl. Uniform Resource Locator) - Enolični krajevnik vira ki predstavlja naslov spletne strani v svetovnem spletu.

CRUD - (angl. create-read-update-delete) - Kratica predstavlja štiri osnovne operacije nad bazo podatkov.

SPA - (angl. single page application) - Je vrsta spletne aplikacije, ki se nahaja na eni strani z namenom bolj tekoče uporabniške izkušnje.

Povzetek

Uporabniki svetovnega spleta se dnevno soočamo z iskanjem informacij. Pri iskanju le teh so nam v veliko pomoč spletni iskalniki, ki nam ob vnosu poizvedbe vrnejo številne rezultate. Kljub temu da so ti zadetki odlično izbrani, moramo za iskanje relavantne informacije pregledati število strani. V diplomskem delu sem se ukvarjal z reševanjem tega problema.
Razvil sem spletno aplikacijo, ki združuje informacije, pridobljene iz različnih domen. Na nadzorni plošči ima tako uporabnik enoten vpogled nad odrezki, ki predstavljajo dele iz različnih strani. Odrezke lahko uporabnik dodaja preko razširitve ali pa jih ureja. Nastavi si lahko tudi alarme, preko katerih ga sistem obvešča o spremembah, ki se zgodijo.
Sistem sem realiziral s pomočjo spletnih pajkov, ki v ozadju aplikacije luščijo podatke in jih shranjujejo v podatkovni bazi. Aplikacija potem te podatke na podlagi pravil predeluje v obvestila.

Ključne besede: pajek, luščenje, aplikacija, servis, razširitev, obvestilo.

Abstract

Web users are searching information on the internet on daily basis. Easiest way to acquire data from the internet is by using search engines, that provide us with many different results. Despite the fact these results are being well chosen for us, there is still a great deal of filtering involved when looking for vital information. In my thesis I have also dealt with solving this problem myself. I wrote a web application in which I merged the information from several domains. The application's dashboard enables the user a complete overview of snippets from various websites. The user can add and edit the snippets themselves. One can set up alarms through which the system informs the user of changes that have occurred. I implemented the system with the help of web crawlers that scrape data and saves it to a database.

Keywords: spider, scraping, application, servis, extension, notification.

Poglavje 1

Uvod

V zadnjem času se na spletu pojavlja veliko spletnih aplikacij, ki agregirajo podatke določene domene (npr. nepremičnine, hoteli, letalske karte, ...) ter ponujajo enoten vpogled. S tem prihranimo čas, ki bi ga sicer potrebovali za obiskovanje posamične spletne strani. Agregirane podatke lažje uredimo in primerjamo med seboj neodvisno od vira , od koder prihajajo.

Iz te opazke izhaja tudi ideja za mojo diplomsko nalogo. Raziskoval sem principe in metode luščenja podatkov iz različnih spletnih strani. Te podatke lahko pridobimo na več načinov. Osredotočil sem se predvsem na luščenje podatkov s pomočjo pajkov. Tako je bil moj cilj razviti spletno aplikacijo, ki bo omogočala tako zajem podatkov iz drugih spletnih strani in portalov, kot tudi pripravo pravil, ki povedo, ob kakšnih dogodkih (spremembah ali podatkih na posameznih spletnih straneh) bi želel uporabnik biti o tem obveščen. Na primer uporabnik bi želel dobiti SMS obvestilo, če je za področje Ljubljane napovedano deževno vreme za več kot dva dni (na portalu Meteo) ter zapora na Slovenski cesti (na portalu Promet.si).

V prvem poglavju bom govoril o splošnih pojmih, ki se bodo pojavljali v nadaljnih delih diplomske nalog in o tehnologiji, ki sem jo uporabil za izvedbo in razvoj. V drugem delu bom predstavil idejno zasnovo in načrt rešitve, nato nadaljeval s samo implementacijo sistema in na koncu naredil analizo ter predstavil možne nadgradnje.

Poglavje 2

Osnovni pojmi in tehnologije

2.1 Semantični splet

Splet, kot ga poznamo danes, deluje na principu zahtev in odgovorov. Ko v spletni brskalnik vnesemo povezavo do spletne strani, ta pošlje zahtevo na strežnik, kjer strežnik sprocesira zahtevo in odgovori z dokumentom. Brskalnik prejme dokument in ustvari model s strukturo drevesa. Na vrhu modela imamo objekt tipa `document`, ki predstavlja celoten dokument. Na koncu brskalnik prikaže stran s pomočjo stroja za upodabljanje (angl. rendering engine) [1]. Tako smo računalnike povezali v splet, kjer si med seboj izmenjujejo podatke.

"Semantičen splet predstavlja delovni okvir, ki omogoča deljenje in ponovno rabo podatkov med aplikacijami, družbami in skupnostmi," je leta 2001 zapisal britanski informatik Tim Berners-Lee [2]. Semantični splet je splet, na katerem so podatki predstavljeni tako, da jih razume tudi stroj. Če bi računalnik ob procesiranju prejete spletne strani razumel, da se nek del strani nanaša na dogodek in bi znal razbrati kraj, čas in tip dogodka, bi na primer lahko preveril v koledarju uporabnika, ali je takrat prost in mu predlagal obisk dogodka. Ko predpostavimo semantiko, pridemo do veliko izboljšav. Dodatno dimenzijo dobijo spletni iskalniki, ki podatke iščejo tudi na podlagi konteksta, lokacije in sopomenk ter ne le na podlagi ključnih be-

sed. Taki rezultati so bolj relavantni. Semantično iskanje danes uporabljajo vsi večji iskalniki, kot so Google, Bing in Yahoo.

2.2 Spletno luščenje podatkov

Eden od izzivov navadnega spleta je tudi, kako izluščiti relevantne podatke iz posamezne spletne strani. Če me na primer zanima vremenska napoved za Ljubljano za prihajajoči vikend, lahko seveda te podatke dobim, če obiščem spletni portal vreme in vpišem ustrezne parametre. Vprašanje pa je, ali lahko računalniški program sam poišče te podatke? Poznamo več vrst luščenja podatkov [3]:

Ročni zajem — Pri spletnem luščenju se lahko zgodi, da niti najboljši algoritem ne reši težave. Takrat nam ne preostane drugega, kot da ročno kopiramo in prilepimo vsebino.

Luščenje z uporabo regularnih izrazov — Vsebino lahko iz spletnih strani luščimo z uporabo regularnih izrazov, ki so vključeni v številnih orodjih. Ponujajo jih tudi nekateri brskalniki.

HTTP programiranje — Programi lahko pošljejo zahtevo po dokumentu preko vtičnika. Ko dokument prejmejo, lahko iz njega luščijo podatke.

HTML razčlenjevalniki — Veliko strani se prikazuje dinamično na podlagi podpornega sistema, kot je podatkovna baza. Poizvedovalni jeziki, kot sta XQuery in HTQL, so sposobni poizvedovanja po spletnih straneh in vračajo vsebino.

Razčlenjevanje DOM strukture — Spletni brskalniki, kot so Google Chrome, Mozilla Firefox, Edge in Safari nam dovoljujo luščenje preko integriranih odjemalnih skript ali preko razširjevalnikov.

Uporaba aplikacij za luščenje spletnih strani — Obstajajo aplikacije, ki same prepoznajo strukturo spletne strani in zgenerirajo algoritem za luščenje podatkov.

Uporaba motod vertikalne agregacije — Metoda je specifična za določena podjetja.

Prepoznavanje semantičnih anotacij — Tehnika preuči spletno stran preko anotacij in meta podatkov, ki so del HTML dokumenta , in jih pišejo razvijalci spletnih strani. Tako zgradi strukturo, preko katere potem izlušči podatke.

Luščenje podatkov s pomočjo računalniškega vida — Pri tem načinu algoritmi luščijo podatke preko grafičnega zajema prikazovalnika.

Možnosti uporabe luščenja podatkov je res veliko. Za primer vzemimo podjetje, ki si želi nadzirati cene svojih konkurentov, ali pa popotnika, ki si želi nakupa najcenejše letalske karte, brez da bi mu bilo potrebno obiskati številne spletne strani in agencije. Oba lahko obiščeta eno stran, ki najde najboljše rezultate. Podjetje se lahko obrne na Media tool kit [4], popotnik pa svojo letalsko karto poišče na strani Skyscenner [5].

2.3 Pajek

Če lastnik spletne strani ne ponudi uradnega aplikacijskega vmestnika, preko katerega bi lahko avtomatsko dostopali do podatkov, ki jih objavlja na spletni strani, nam ne ostane drugega, kot da napišemo tako imenovan "web crawler". To je program, ki na podlagi podanega vzorca izlušči željene podatke. Drugo ime za "web crawler" je bot ali pajek, katerega bom uporabljal v nadaljevanju. Večina pajkov na spletu, je napisanih predvsem z namenom indeksiranja [6]. Večji spletni iskalniki (angl. search engines), kot je Google, uporabljajo indeksiranje za zbiranje podatkov o samih spletnih straneh. Googlebot [7], Googlov glavni pajek za indeksiranje, tako pošlje HTTP zahtevo na nek URL in iz meta značk v dokumentu HTML izlušči informacije o tej strani. V meta značkah imamo tako definiran naslov, opis in ostale parametre, ki pomagajo ali preprečujejo pajkom dostop do informacij. Med bolj pomembnimi je meta značka `robots` ali `googlebot`, v kateri lahko

preprečimo ineksiranje, poglabljanje, prikaz odrezkov (angl. snippet), arhiviranje in indeksiranje slik. Zraven meta značk, ki služijo bolj indeksiranju, morajo vsi pajki upoštevati tudi datoteko `robots.txt` [8]. Tako naj bi pajki najprej odprli to datoteko, ki jim služi kot nek bonton. Do nje pridemo tako, da spletni strani na koncu korenske domene strani dodamo `/robots.txt`. Če imamo na strani vsebino, ki nočemo, da je vidna pajkom, ali pa želimo definirati posebna pravila, lahko to zapišemo v `robots.txt`.

```
User-agent: *
Disallow: /videos
Allow: /videos/seeside.mp4
```

User-agent: Določa, za kakšne vrste pajkov veljajo pravila. V našem primeru imamo postavljeno zvezdico, kar pomeni, da pravila veljajo za vse pajke. Če bi namesto zvezdice napisali `Googlebot`, bi to pomenilo, da pravilo velja le za agenta Googlebot.

Disallow: Je del, ki pajku pove, do katerih datotek naj ne bi dostopal. Pajek bi naj tako ignoriral URL `/videos`.

Allow: Definira izjeme glede na pravila, ki so definirana v sklopu `Disallow`. Pajek tako lahko indeksira video posnetek `seeside.mp4`, čeprav je v mapi `/videos`.

Pajek je seveda algoritem, ki ga navadno zaženemo na strežniku. Pri luščenju podatkov moramo biti pozorni predvsem na to, da je odporen na strukturo značk spletne strani - kar pomeni, da tudi če lastnik spletne strani spremeni malenkosti označevalnega jezika, pajek še deluje.

2.4 DOM

DOM ali objektni model dokumentov je jezikovno neodvisno računalniško okolje, ki omogoča programom in skriptam dinamični dostop ter spremembo

strukture in stila dokumentov. Te lahko asinhrono osvežujemo ali pri novejši verziji DOM 3 tudi dodajamo. Na primer v neurejen seznam dodamo element. Struktura dokumnetov je urejena na podlagi drevesa, kjer vsako vozlišče predstavlja objekt. Preko programskega jezika Javascript imamo tako na primer možnost pregleda celotne HTML strukture dokumneta. HTML dokument je možno pokazati v obliki drevesa, kjer ena značka predstavlja vozlišče in ima sinove ter starše. Če vzamemo značko `<header>` iz primera spodaj, vidimo, da ima sina `<ul>` in starša `<body>`.

```
<body>
    <header>
        <ul>
           <li>Prvi element neurejenega seznama</li>
           <li>Drugi element</li>
        </ul>
    </header>
</body
```

Preteklost modela seže nazaj vse do razvoja prvega spletnega brskalnika Netscape in konkurenta IE. Obe podjetji sta razvili tudi prvi verziji programskega jezika Javascript in JScript. Tako Netscape kot IE sta čez eno leto predstavila svoji verziji objektnega modela dokumentov. Po ustanovitvi W3C, je standarde modela DOM, katerega poznamo danes, zapisala organizacija W3C, ki še danes skrbi za njegov razvoj in napredek.

2.5 Tehnologije

2.5.1 Python

Python [9] je visokonivojski objektno usmerjen programski jezik, zgrajen pod odprtokodno domeno. Veliko programerjem, med katere spadam tudi sam je všeč prava kombinacija[10] zadostne hitrosti, precejšnje razširjenosti, veliko množico knjižnic (pribl. 66, tisoč) [11] in predvsem pregledenost. S

svojo sintakso so ustvarjalci izločili veliko nepotrebnih zavitih oklepajev in jih nadomestili s presledki. Python je tolmačitveni programski jezik, kar pomeni, da računalnik izvaja progamsko kodo, zapisano v visokonivojskem jeziku, in je ne prevaja v strojni jezik. Tolmačenje programske kode nam pomaga pri hitrejšem razvoju, kjer preskočimo korak prevajanja. Slaba stran tega se opazi pri končni hitrosti algoritma, ki je tako manjša kot pri programskih jezikih, kot je C [12].

V programskem jeziku Pyhton sem poleg programskih ogrodij Django in Scrapy uporabljal tudi naslednje knjižnice:

- pip,
- virtualenv,
- virtualenvwprapper,
- mysql-connector-python,
- Pillow,
- scrapy-djangoitem,
- Twisted,
- Scrapyd,
- Selenium.

2.5.2 HTML

Označevalni jezik HTML se je pojavil že s samim začetkom spleta. Predstavlja ga široka paleta značk, ki jih programer uporablja pri razvoju spletne strani. Te značke so na primer glava, telo, naslov, podnaslov, slika itd. Kot že rečeno so značke urejene v drevo, ki je tvorjeno iz HTML dokumenta. Značka tako predstavlja eno vozlišče in lahko vsebuje besedilo ali atribut. Med pomembnejše atribute bi lahko šteli `class` in `id`, ki služita kot sistem

za poimenovanje značk. Uporabjena sta predvsem za določanje stila elementov strani ali manipulicijo z Javascriptom.

Od začetka ustanovitve organizacije W3C ti skrbijo za standardizacijo in razvoj označevalnega jezika HTML. Danes je tako na spletu aktualna verzija HTML5 [13], ki je izšla oktobra 2014. Ponuja nam razširjeno vrsto značk, kot so `<video>, <audio>, <footer>, <header>, ...` in spremembe med atributi. Z dodatnimi značkami, ki jih najdemo v označevalnem jeziku HTML5, so razvijlci izboljšali semantičnost spleta.

2.5.3 CSS

CSS ali "Cascading Style Sheet" je mehanizem za dodajanje stila označevalnem jeziku HTML. Jezik je bil razvit z namenom ločitve stila s samim ogrodjem, ki ga razvijemo s pomočjo označevalnega jezika HTML. Vključimo ga lahko na tri načine: direktno z atributom `style`, neposredno preko značke `<style>` ali pa preko zunanje datoteke. Če ga uporabljamo preko zunanje datoteke, ima ta končnico `css`. V dokumentu s pomočjo enoličnih identifikatorjev (angl. id) in razredov (angl. class) ciljnim značkam definiramo stil. Če znački določamo stil preko enoličnega identifikatorja, ta velja striktno za njo, medtem ko razred cilja vse elemente, ki mu pripadajo. CSS nam omogoča tudi naprednejše načine izbire elementov. Na primer če uporabimo presledek med dvema razredoma `.razred1 .razred2 {}` , s tem ciljamo vse sinove prvega razreda, ki vsebujejo drugi razred.

Trenutno je na voljo že tretja različica jezika CSS, ki nam med drugim ponuja tudi animacije in preprosto 3D oblikovanje. Za razvoj tega oblikovalnega jezika skrbi organizacija W3C.

2.5.4 Javascript

Je objektini programski jezik, razvit z namenom boljše interaktivnosti spletnih strani. Razvilo ga je podjetje Netscape, ki ga je javno podprlo v drugi uradni verziji svojega brskalnika. Jezik je standardiziran s specifikacijo

ECMAScript [14]. Letos junija je izšla že šesta verzija specifikacije.

Preko Javascripta, ki se izvaja v spletnem brskalniku, lahko pošiljamo oddaljene AJAX klice, animiramo/dodajamo/odstranjujemo elemente, ženemo interaktivne vsebine, validiramo vhodna polja ali vodimo analize. Pri večini teh obnašanj Javascript ob nekem dogodku manipulira z objektnim modelom dokumentov. Dogodke uporabnik sproži s svojimi akcijami v brskalniku.

Javascript je jezik, ki je v zadnjih letih doživel veliko vzpona. S prihodom odprtokodne platforme Node.js se Javascript uveljavlja tudi za pisanje strežniških aplikacij.

2.5.5 jQuery

Odprtokodna knjižnica jQuery [15] je napisana za programski jezik Javascript. Avtor John Resig jo je napisal z namenom preprostejše interakcije med označevalnim jezikom HTML in programskim jezikom Javacript. Knjižnica je podprta na vseh večjih spletnih brskalnikih in uporabljena v približno 30 odstotkih vseh spletni strani [16]. Za podporo ciljanja in manipulacijo nam služi glaven objekt jQuery, do katerega dostopamo z znakom `$` ali z nizom `jQuery`. Omogoča nam tudi ciljanje njegovih sinov, naslednjikov, predhodnikov, staršev ali vseh sorodnih elementov hkrati.

```
$( document ).ready(function() {

$("img").hover(function(){
        $('#glava').show();       (1)
    },function(){
        $('#glava').hide();        (2)
    });

});
```

Zaradi oddaljenih klicov, ki jih brskalnik pošilja strežniku, pri izmnejavi

dokumnetov preteče nekaj časa, preden je DOM pripravljen za izvajanje Javascripta. Ker se želimo izogniti praznemu ciljanju elementov, je pomembno, da s funkcijo `ready()` počakamo, da se dokument naloži do konca.

V premeru ciljamo značko `<img>`, ki v HTML jeziku predstavlja sliko. Ob premiku miške na sliko se izvede prva akcija (1), kjer izberemo element z identifikatorjem `glava` in ga prikažemo na spletni strani. Za zaznavanje premika miške na sliko jQuery uporablja poslušalca dogodkov, imenovanega `mouseover`. Knjižnica v tem primeru tako na začetku vsem slikam doda poslušalca dogodkov. Ko miško premaknemo iz slike, se proži sekundarna akcija, ki skrije element z identifikatorjem `glava`. Za zaznavanje tega dogodka skrbi poslušalec `mouseout`, ki se proži, ko miška zapusti trenutno sliko.

JQuery nam zraven lažjega dela z dogodki ponuja še lažje pošiljanje klicev AJAX. Tako lahko že z nekaj vrsticami kode pošljemo POST, GET ali PUT zahtevo na določen naslov.

2.5.6 XPath

"XML Path Language" ali XPath[17] je poizvedovalni jezik, namenjen naslavljanju delov XML dokumenta. V podporo primarnemu namenu XPath zagotavlja tudi osnovne operacije manipuliranja z nizi, števili in logičnimi operatorji. XPath deluje na abstraktnem modelu logične strukture XML dokumenta.

2.5.7 Scrapy

Je odprtokodno delovno ogrodje za pisanje pajkov in procesiranje pridobljenih podatkov. Scrapy je napisan v programskem jeziku Python, kar nam omogoča lahko razširitev njegove funkcionalnosti z vključitvijo pythonskih knjižnic [18]. Ko si na računanik namestimo delovno okolje in zaženemo nov projekt, se nam v izbrani mapi vzpostavi struktura datotek, prikazana spodaj.

```
primer/
```

```
scrapy.cfg
primer/
    __init__.py
    items.py          (2)
    pipelines.py      (3)
    settings.py       (4)
    spiders/
        __init__.py
        vreme.py      (1)
```

1. V podmapi spiders se nahajajo pajki (v zgornjem primeru `vreme.py`). Pri pisanju pajka moramo najprej definirati podrazred, ki razširja razred `scrapy.Spider` in določiti atribute `name`, `start_urls` in metodo `parse()`. Prvi atribut `name` določa ime pajka in mora biti enoločen. Drugi atribut `start_urls` je seznam, v katerem naštejemo začetne povezave, ki jih obišče naš pajek. Metoda `parse()` definira premikanje po straneh, izbiranje elementov in zajemanje podatkov. Kot vhodni argument sprejme objekt `response`, ki ga vrne stran, definirana v seznamu `start_urls`. Metoda `parse()` vrne objekt `Item`.

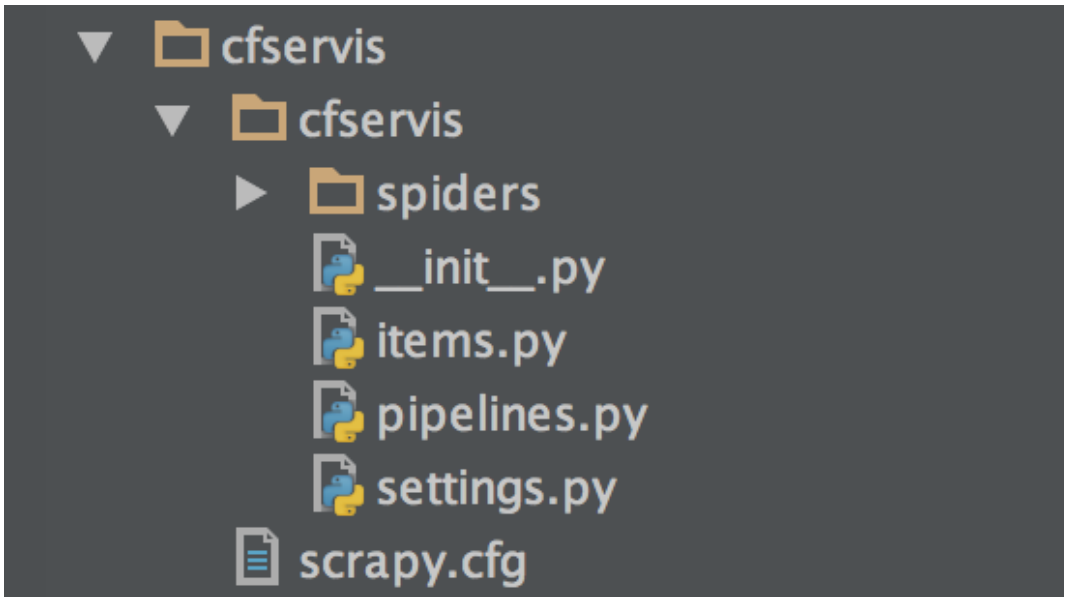
2. V datoteki definiramo razrede, ki določajo obliko izluščenih podatkov.

3. Ko preko skripte ali terminala z ukazom `crawl` zaženemo pajka, ta obšče podan naslov, vrne objekt `Item` in ga pošlje v cevovod (angl. pipeline). Glavna naloga cevovoda je nadaljna obdelava objekta, ki mu ga vrne pajek. Podatke tukaj pošljemo naprej in jih na primer shranimo v datoteko ali v podatkovno bazo.

4. Datoteka definira glavne lastnosti projekta Scrapy.

2.5.8 Django

Je delovno ogrodje, spisano v visokonivojskem programskem jeziku Python, namenjeno izdelavi spletnih strani. Svoje ime je dobil po znanem jazz ki-



Slika 2.1: Struktura datotek v projektu Scrapy.

taristu iz devedesetih, razvijalci pa ga opisujejo kot "hiter razvojni cikel in manj kode" [19]. Django implementira sistem, ki omogoča ponovno uporabo delov kode. Če imamo podjetje, ki se ukvarja z izdelavo spletnih trgovin, nam Django omogoča izdelavo ogrodja, ki ga kasneje lahko uporabimo pri vsaki naslednji spletni trgovini, ki jo izdelujemo. Velika prednost delovnega ogrodja Django je tudi njegova arhitektura. Njegov MVC sistem med seboj loči model, pogled in kontroler.

Model definira obliko objektov, ki jih shranjujemo. Te razrede običajno definiramo v datoteki `models.py` in razširjajo razred `models.Model`. Django uporablja ORM [20], kar nam omogoča uporabo objektnega poizvedovanja nad podatkovno bazo. Zaradi preslikovanja objektov v entitete in obratno je pomembna sinhronizacija modelov s strukturo podatkovne baze, definirane na podatkovnem strežniku (npr. s strukturo relacijske podatkovne baze MySQL).

Pod besedo pogled (angl. view) pri ogrodju Django mislimo predloge (angl. templates). Ker je cilj MVC-ja ločitev različnih delov kode, so raz-

vijalci ogrodja Django razvili poseben jezik Django Themplate Language, ki se uporablja v predlogah. Koda se nahaja v datotekah s končnico `html`, in jo Django izvede, preden dokument pošlje k uporabniku.

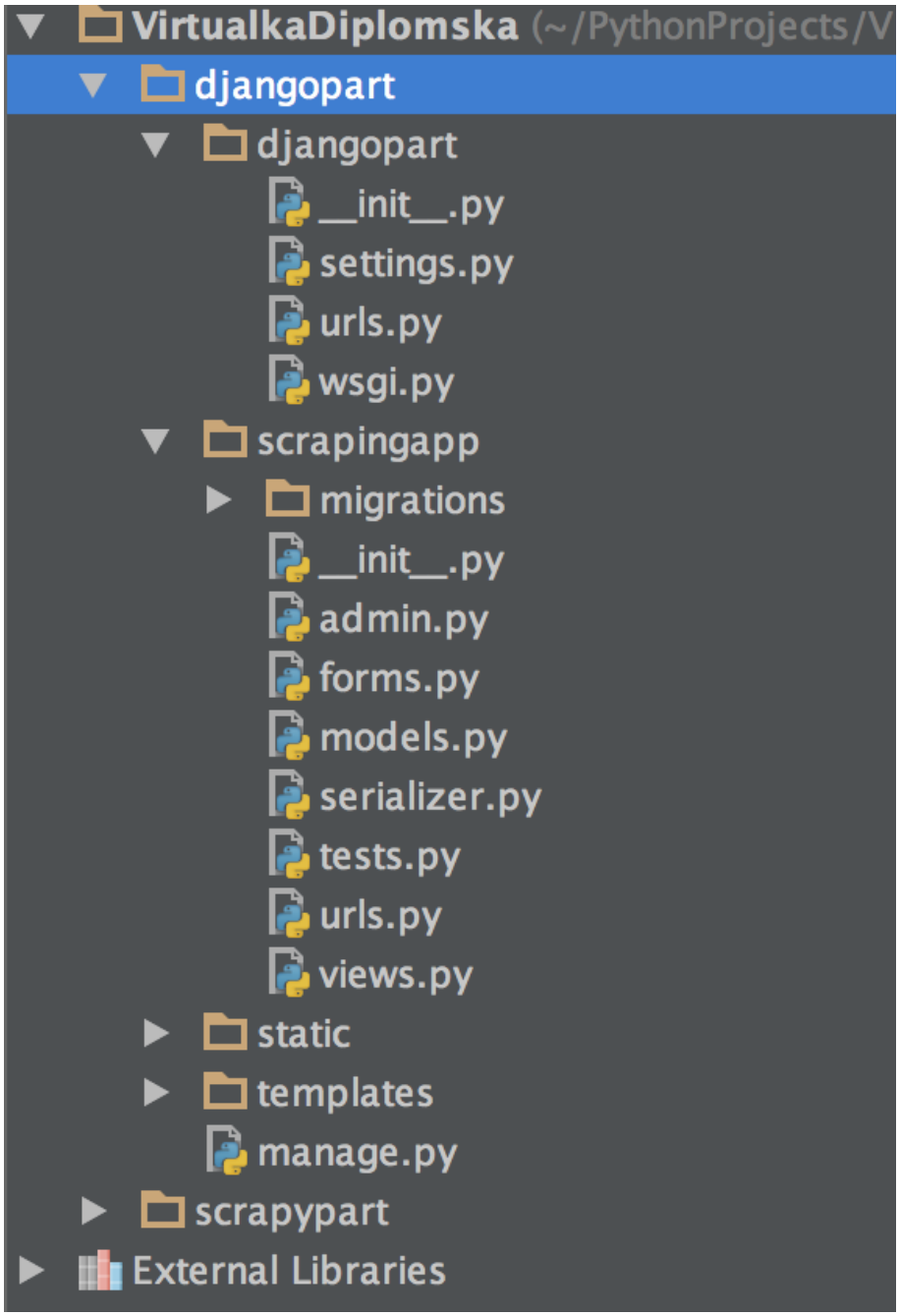
Kontroler (angl. controller) je del, v katerem se nahaja poslovna logika in služi kot vez med uporabnikom in podatkovno bazo. Zapisan je v datoteki, imenovani `views.py`.

Z ukazom `django-admin startproject` ustvarimo novo instanco projekta. Če gledamo Sliko 2.2, je to zunanja mapa z imenom djangopart. V njej se nahajajo datoteke:

- `manage.py` — Datoteka, namenjena manipulaciji projekta preko ukazne vrstice.
- `__init__.py` — Služi kot konstruktor.
- `settings.py` — Hrani osnovne nastavitve projekta, kot so povezava z podatkovno bazo, časovni pas, seznam inštaliranih aplikacij in različne poti do datotek.
- `urls.py` — V ogrodju Django imamo več nivojsko strukturo povezav, na katere uporabnike pošiljajo zahteve. Tukaj tako ponavadi vključimo datoteko s povezavami iz aplikacije ali pa povezavo definiramo kar direktno.
- `wsgi.py` — Je vstopna točka za strežnike, ki podpirajo WSGI. Npr. pri postavitvi (angl. deployment) projekta na strežnik v datoteki določimo potrebne usmeritve.

Django projekt nam sam od sebe ne nudi ničesar drugega kot osnovno plast. Če želimo razviti spletno stran, moramo zagnati še ukaz `startapp`, ki nam ustvari aplikacijo znotraj projekta. V tej aplikacije se nahajajo naslednje datoteke:

- `admin.py` — V datoteki določimo, kateri objekti so dosegljivi preko Djangojevega administrativnega uporabniškega vmesnika.



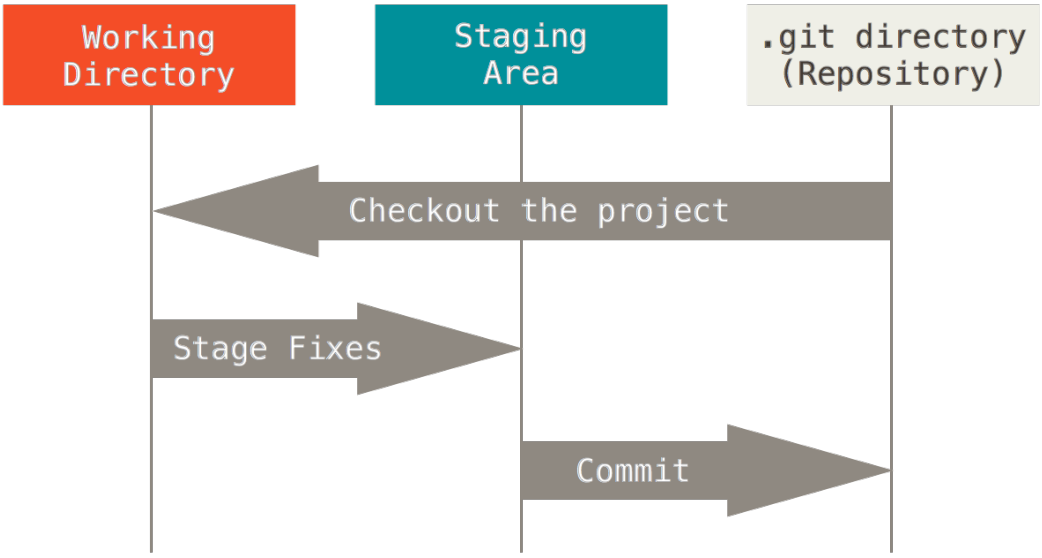
Slika 2.2: Struktura datotek v projektu Scrapy.

- `models.py` — V datoteki se nahajajo razredi, ki predstavljajo objektno sliko podatkovne baze.
- `views.py` — Tukaj se nahajajo pogledi, preko katerih strežnik servira podatke, pobrane iz podatkovne baze, in jih preko odgovora pošlje uporabniku. Večina pogledov na koncu kliče metodo `redner()`, ki sprejme podatke, in dokument HTML ter vrne objekt tipa `HttpResponse`.
- `urls.py` — Datoteka je nemenjena povezavam, ki se navezujejo na to aplikacijo. Eno povezavo predstavlja ena vrstica. Predstavljena je z metodo `url()`, ki ji lahko nastavimo štiri argumente. Prvi je sestavljen iz regularnih ukazov, ki služijo kot šablona za preverjanje naslova. Če se naslov ujema z regularnim izrazom [21], nas Django preusmeri na določen pogled, ki je definiran v drugem argumentu. Pogledu lahko pošljemo tudi vrednosti, za to je na voljo tretji argumnet `kwargs`. Četrti argument pa predstavlja ime pogleda.
- `tests.py` — V njej napišemo teste enote, ki služijo razhroščanju programskih napak.

2.5.9 Git

Git je VCS — sistem za nadzor različic programske kode [22]. Sistem je bil razvit leta 2005 s strani švedskega razvijalca programske opreme Linusa Torvaldsa. Prvo je bil uporabljen za nadzor različic jedra Linux. Git je izjemno hiter in sedaj uporabljen pri večini večjih projektih. Za razliko od drugih sistemov Git ne shranjuje skupka datotek in sprememb med njimi, ampak za vsako oddajo naredi sliko. Tako datoteke ne shranjuje znova, ampak ustvari le referenco na datoteko, ki jo ima že shranjeno. Svoje podatke Git interpretira bolj kot tok podatkov.

Git shranjuje svoje dokumente lokalno in oddaljeno. Ker so datoteke shranjene lokalno, za nadaljno delo na projektu ne rabimo VPN povezave s strežnikom. Git pozna tri stanja, v katerih se nahaja projekt: delovni direktorij, git direktorij in vmesno področje.



Slika 2.3: Delovni direktorij, vmesno področje in Git direktorij.

Git direktorij je najpomembnejši del [23]; tu Git shranjuje meta podatke in objekte našega projekta.

Delovni direktorij je prostor, kjer hranimo datoteke, ki jih konstantno spreminjamo in dopoljnjujemo. Te datoteke so prenesene iz zgoščene podatkovne baze Git direktorija.

Vmesno področje (angl. Staging Area) je datoteka, v kateri hranimo informacije o dokumentih, ki bodo zavzeti v naslednjem izročanju (angl. commit).

Preprosti tok podatkov deluje tako, da datoteke spreminjamo v delovnem direktoriju. Tako jih Git označi kot spremenjene (angl. modified). Če to datoteko potem dodamo še v vmesno področje, postane `staged`. Datoteke, ki so označene kot `staged`, lahko potem izročimo z ukazom `commit`.

2.5.10 Google Chrome razširitve

Leto po prihodu spletnega brskalnika Chrome je Google razvil tudi ogrodje za razširitve [24]. Namen teh je spremeniti in izboljšati osnovne možnosti,

ki jih ponuja sam brskalnik. Razširitve so napisane v spletnih tehnologijah Javascript, HTML in CSS, zato so precej podobne spletnim stranem in lahko uporabljajo aplikacijske vmestnike, ki jih implementira spletni brskalnik (XMLHttpRequest, JSON, HTML5) [25]. Komunicirajo lahko s spletnimi stranmi, v njih vključijo skripte in upravljajo z zavihtki ali zaznamki.

Vsaka razširitev vsebuje datoteko `manifest`, eno ali več HTML datotek, eno ali več Javascript datotek in dodatne datoteke kot so slike, zvoki in videoposnetki. Datoteke hranimo v mapi, ki jo kasneje pri postavitvi (eng. deployment) na spletno trgovino Google Store [26] shranimo v arhivu ZIP s končnico `.crx`.

V datoteki `manifest`, ki je tipa JSON, določimo osnovne informacije o razširitvi, zmogljivost delovanja in pomembnejše datoteke.

2.6 Razvojna orodja

2.6.1 PyCharm

PyCharm je razvojno orodje za celovit razoj aplikacij, napisanih v programskem jeziku Python. Aplikacija vsebuje vrsto naprednih funkcionalnosti, kot so inteligentni urejevalnik besedila, grafični razhroščevalnik, sistem za verzioniranje in podpora Pythonovem delovnemu ogrodju Django. Podjetje JetBrains češkega porekla je program PyCharm razvilo leta 2010. Od takrat je orodje dobilo veliko posodobitev.

2.6.2 Brackets

Je razširljivo odprtokodno programsko orodje, ki nastaja pod okriljem podjetja Adobe. Programsko orodje je še mlado, prva verzija je bila izdana komaj lani novembra. Kljub temu izstopa na trgu zaradi treh stvari [27]: ker je preprostejša — okolje ti ne prikazuje nepotrebnih okenc, ki jih v tem trenutku ne potrebuješ, sinhronizinan je z brskalnikom — kar nam omogoča takojšnji pregled sprememb, ki jih pišemo v programski kodi, in ker je zapi-

sana v spletnih tehnologijah — tako imajo razvijalci lažjo možnost dodajanja in spreminjanja urejevalnika po svoje.

2.6.3 Google Chrome Canary

Google Chrome je spletni brskalnik, razvit s strani podjetja Google pod odprtokodnim projektom Chromium. Prva verzija je izšla leta 2008 za Microsoftova Okna v 43 jezikih. Chrome uporablja sistem za upodabljanje (angl. rendering), imenovan Blink, ki je baziran na WebKitu [28]. Danes je to moderni spletni brskalnik s kar 64,8 odstotnim deležem trga [29].

Pri razvoju spletne aplikacije velikokrat naredimo napako v programski kodi. Pri razhroščanju te napake nam je v veliko pomoč Chromov način za razvijalce. Če skripto pišemo v programskem jeziku Javascript, potem si lahko npr. v konzoli izpisujemo napake in testiramo delovanje skript. Način za razvijalce nam služi tudi za analiziranje upodabljanja (angl. rendering), pregled strukture DOM [30] in emulacije različnih prikazovalnikov (angl. screen).

Google Chrome Canary je procesna veja, ki se z brskalnikom Chrome razvija sočasno. Prednost veje Canary je v dnevnih posodobitvah, ki jih uporabniki dobijo vsak večer. Ker se verzija posodablja dnevno, se zgodi, da vsebuje več hroščev, kar lahko posledično pomeni, da se brskalnik občasno zruši.

2.6.4 Source Tree

Source Tree je odjemalni program, ki zamenja uporabo Gita preko ukazne vrstice z grafičnim vmesnikom. Preko njega lahko upravljamo s predhodno ustvarjeno shrambo podatkov ali pa si ustvarimo novo. Ko izberemo Git shrambo, imamo pregled nad vsemi oddajami, grafični pregled dokumentov, ki se nahajajo v vmesnem območju, in ukaznimi gumbi, ki služijo upravljanju s celotno shrambo. Po mnenju strokovnjakov [31] je Source Tree dober prav zaradi dobrega pregleda nad orodjem Git. Na preprost in inovativen način

združi vse v svojem vmesniku.

2.6.5 Terminal

Aplikacija Terminal je posnemovalnik terminala. Ta del programske opreme je vključen v operacijskegem sistemu OSX, ki ga razvija podjetje Apple. Posnemovalnik terminala nam nudi tekstovni nadzor nad operacijskim sistemom z uprabo ukazne vrstice. Ukaze pišemo v jeziku nadzornika poslov, imenovanem Bash.

Poglavje 3

Pregled obstoječih rešitev

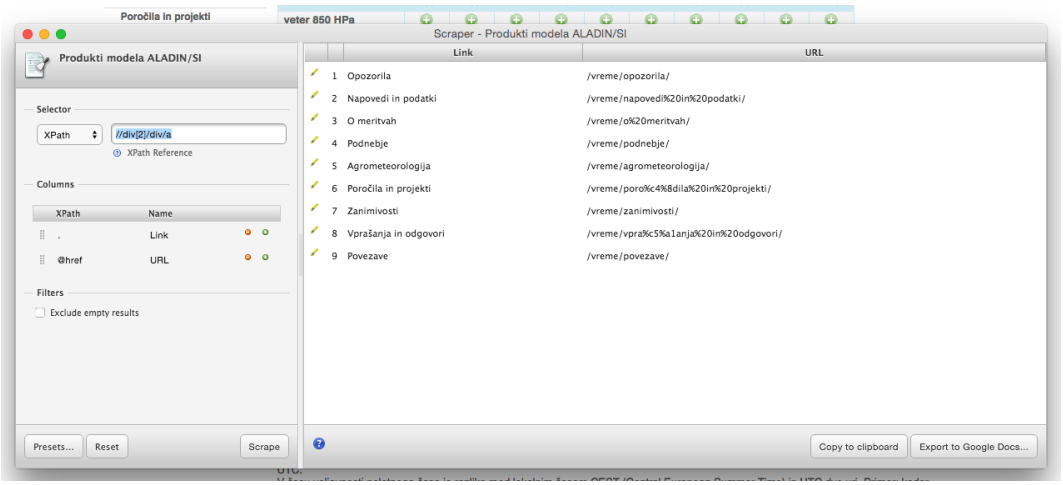
Obstaja veliko rešitev, ki rešujejo problem luščenja podatkov iz spletnih strani. V tem poglavju sem izpostavil tiste, ki rešujejo podobno kot moja aplikacija.

3.1 Razširitev Scrapy

Scrapy je razširitev za spletni brskalnik Google Chrome, ki si jo lahko namestimo preko spletne strani Google Web Store [26]. Deluje tako, da si najprej z desnim klikom izberemo element in razširitev nam na podlagi vzorca sama vrne podobne elemente. Dobljene elemente lahko potem izvozimo v Google Docs [32] ali shranimo v odložišče. Delovanje je prikazano na Sliki 3.1.

3.2 Safari web clip widget

V operacijskem sistemu OSX [33] si lahko uporabniki preko tega gradnika Safafi web clip widget [34] shranijo del spletne strani v svojo nadzorno ploščo (angl. dashboard). Gradnik se osveži vedno, ko se na strani zgodi sprememba.



Slika 3.1: Prikaz delovanja razširitve Scrapy.

3.3 Firebug

Firebug je odprtokodni gradnik spletnega brskalnika Firefox. Vsebuje veliko različnih komponent, ki so v podporo razvijalcem spletnih rešitev. Ena izmed komponent je razčlenjevanje strukture DOM. Z razčlenjevanjem tako pridemo tudi do željenih podatkov.

3.4 Mediatoolkit

Mediatoolkit je orodje za spremljanje relavantnih omemb podjetja na spletu v realnem času. Ob omembi nam orodje pošlje tudi obvestilo.

3.5 Primerjava

Spletna aplikacija, ki sem jo razvil je na prvi pogled podobna gradniku Web clip [34]. Prva razlika je v večji dinamičnosti, ker lahko do odrezkov dostopamo preko brskalnika in nismo omejeni le na nadzorno ploščo, ki jo ponuja operacijski sistem OS X. Druga pomembnejša razlika pa je v dodatnem sistemu za obveščanje. Spletna aplikacija uporabniku omogoča izbiro pravila,

preko katerega mu potem streže obvestila. Podatke sem pridobil z uporabo poizvedovalnega jezika XPath in preko uporabe CSS klasifikatorjev. Podoben način luščenja implementirata tudi gradnika Firebug in Scrapy, le da delujeta samo na trenutno izbrani strani. Za razliko od njiju moja aplikacija podatke lušči iz večih strani, tako deluje tudi Mediatoolkit.

Poglavje 4

Idejna zasnova in načrt rešitve

Uporabniku se po prijavi na domači strani prikažejo odrezki ali delčki drugih spletnih strani. Te odrezke lahko uporabnik dodaja s pomočjo razširitve, ki si jo namesti v brskalnik Google Chrome, ali pa si odrezke izbere na seznamu odrezkov, pripravljenih s strani urednika. Uporabnik lahko tako preko odrezkov, prikazanih na domači strani, spremlja več strani hkrati. V podporo so mu tudi obvestila, ki ga obveščajo o spremembah.

Za primer vzemimo uporabnika, ki se navdušuje nad tehnologijo. Zanimajo ga predvsem novice iz področja mobilne tehnologije, kjer vedno prebere tiste, ki jih napiše moj izbrani avtor. Preko spletne aplikacije si lahko nastavi pravilo, ki mu ob vsakem avtorjevem novem članku pošlje obvestilo. Tako mu sistem omogoči, da nikoli ne izpusti relativnih novic.

Po televiziji vsak dan ob osmih večer vrtijo film. Uporabnik bi si film rad ogledal le, če je žaner drama. Preko spletne aplikacije si lahko na svojo nadzorno proščo doda odrezek sporeda in nastavi pravilo, ki ga bo ob predvajanju takega žanra obvestilo.

4.1 Uporabniške zgodbe

Pri razvoju projekta sem najprej določil uporabniške zgodbe (angl. user stories), [35] ki so mi v nadaljevanju služile za podporo in sledenje. Zgodbe

opisujejo akcije, ki jih uporabnik z določeno vlogo želi ali mora izvesti na spletni strani. V aplikaciji ločimo med dvema tipoma uporabnikov: prvi je navaden uporabnik, ki aplikacijo uporablja dnevno, drugo pa je urednik, ki skrbi za vsebino strani. Uporabniške zgodbe sem zapisal v Tabeli 4.1.

Kot..	si/se želim..	zato..
vsi	prijaviti v spletno aplikacijo	moram biti prej registriran.
uporabnik	si prikaz svojih odrezkov	se moram prijaviti v spletno stran.
uporabnik	dodati odrezek	si moram namestiti razširitev ali odrezek izbrati iz prej definiranih.
uporabnik	odstraniti odrezek	
uporabnik	določiti pravilo	mora biti omogočeno določanje.
uporabnik	prejeti obvsetila	
urednik	pregleda seznama odzrekov	
urednik	omogočiti določanje pravila	mora biti definiran pajek.
urednik	izbirisati odrezek	

Tabela 4.1: Uporabniške zgodbe spletne aplikacije

4.2 Shema sistema

Po določitvi uporabniških zgodb sem poiskal tehnologije, ki najboljše rešujejo zastavljene zgodbe. Najprej sem se reševanja želel lotiti le s programskim ogrodjem Django, ki sem ga uporabil za zaledni sistem, in z nekaj skriptami, napisanimi v Javascriptu, ki bi skrbele za preverjanje pravil in obveščale uporabnika. Naletel sem na težavo. Pri pridobivanju podatkov iz odrezkov, oziroma natančneje pri poizvedovanju po DOM drevesu znotraj značke `<iframe>`, je prišlo do večdomenskega izvajanja kode ali si kratico do XSS (angl. cross site scripting) [36]. Spletni brskalniki imajo vgrajen mehanizem,

ki XSS preprečuje zaradi varnostnih razlogov. Težave sem se kasneje lotil z razvojem storitve, ki preko pajkov izlušči željene podatke.

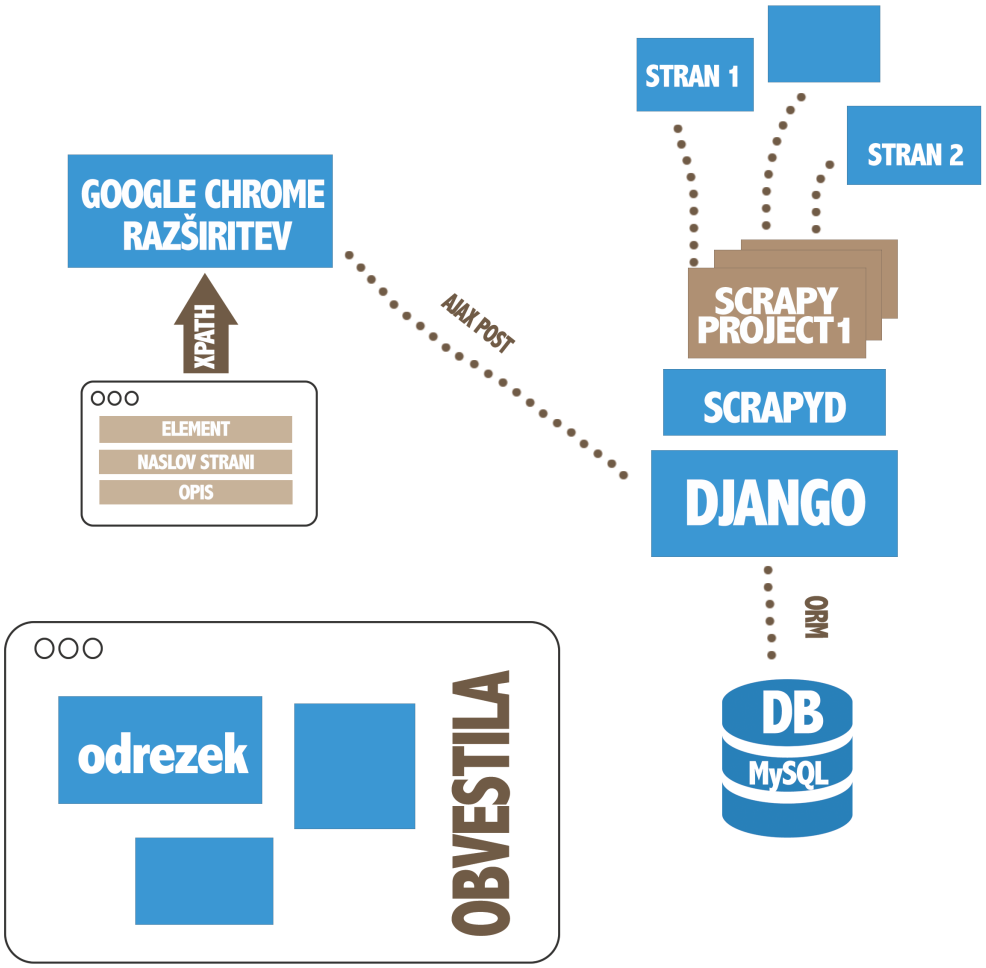
Na Sliki 4.1 si bralec lahko ogleda visoko nivojski prikaz celotnega sistema. Odjemalec pošilja zahteve strežniku, ki je napisan v programskem jeziku Python znotraj delovnega ogrodja Django. Strežnik upodobi odjemalčevo zahtevo, v katero preko poslovne logike vključi podatke. Podatki so shranjeni v podatkovni bazi MySQL. Podatkovna baza in Django sta povezana preko preslikav ORM, kar strežniku omogoča izvajanje CRUD operacij s pomočjo objektov. Če uporabnik želi dodati odrezek na svojo osnovno stran, si preko razširitve izbere element in podatke o njem preko `AJAX POST` zahteve pošlje strežniku.

Sistem uporbnika tudi obvešča preko obvsetil. Za preverjanje sprememb na straneh nam služi servis, napisan v programskem jeziku Python v delovnem ogrodju Scrapy. Servis je postavljen s knjižnico `scrapyd`, ki ima vgrajen tudi API, preko katerega lahko iz serverja Django prožimo zahteve. Ko servis prejme zahtevo na podlagi argumnetov, zažene pajka, ki svoje pridobljene podatke shrani v podatkovni bazi.

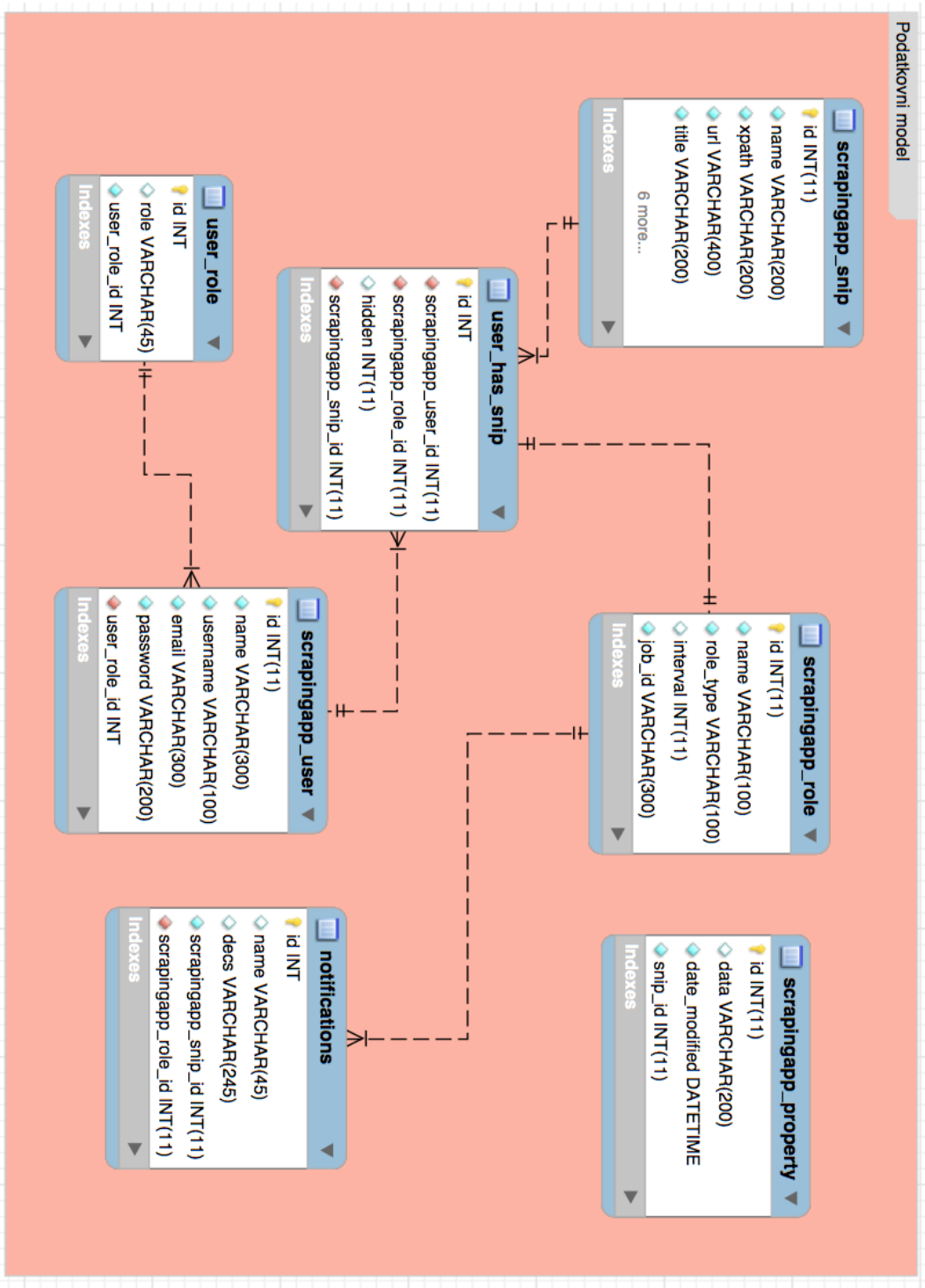
4.3 Konceptualni model podatkovne baze

Podatkovno bazo sem načrtoval s progrmskim orodjem MySQLWorkbench [37], kjer sem si s pomočjo grafičnega urejevalnika narisal konceptualni model podatkovne baze. Ta model sem kasneje preko postopka "forward engineer"preslikal v strukturo podatkovne baze MySQL. Zaradi ORM moramo v ogrodju Django definirati podporne razrede, ki služijo kot podpora preslikavam. Te modele lahko tvorimo z ukazom `python manage.py inspectdb > models.py` .

Ker sem aplikacijo razvijal po agilni metodi[38] razvoja, se je podatkovna baza, prikazana na Sliki 4.2, z napredkom projekta spreminjala/dopolnjevala. Relacije in strukturo sem spreminjal v datoteki `models.py`.



Slika 4.1: Shema celotnega sistema.



Slika 4.2: Konceptualni model podatkovne baze.

Poglavje 5

Implementacija

5.1 Domača stran

Sprednji del aplikacije (angl. frontend) je napisan v označevalnem jeziku HTML 5 in oblikovan z zadnjo različico CSS 3. Za manipulacijo s strukturo DOM in boljšo interaktivnost pa skrbijo skripte, napisane v jeziku Javascript. Na Sliki 5.1 je prikazana domača stran spletne aplikacije. Dinamično se nam prikažejo in osvežujejo odrezki, ki jih lahko tudi skrijemo s klikom na križec znotraj odrezka. Na vrhu strani imamo navigacijsko vrsto, preko katere se lahko premikamo med podstranmi ali prikažemo obvestila. V spustnem seznamu si uporabnik lahko izbere časovni interval, ki služi kot sprožilec dveh stvari:

1. Aplikacija osveži vse prikazane odrezke na domači strani.

2. Pošlje AJAX zahtevek na strežnik. Ob prejemu zahtevka strežnik preveri, če je uporabnik dobil novo obvestilo.

Uporabnik lahko obvestila preveri na vsaki podstrani spletne strani. V desnem kotu pritisne na povezavo `Obvestila` in z desne se pripelje seznam obvestil, kot je prikazano na Sliki 5.2.

Slika 5.1: Prva stran spletne aplikacije.

Slika 5.2: Pregled obvestil.

5.2 Preverjanje novih obvestil

5.2.1 Luščenje novih podatkov

Strežnik pravila preverja ob vsakem prejemu AJAX klica na URL `/posodobi/`. Ob prejetju klica strežnik najprej preveri podatke na vseh straneh s pomočjo podpornega servisa. Podatke je potrebno preveriti na vseh straneh, ki imajo določeno pravilo. Za primer vzemimo pravilo, ki nas obvesti vedno, ko je v Piranu naslednji dan temperatura dovolj visoka za kopanje, npr. 30 stopinj celzij. Pajek za preverjanje temperature na strežnik, kjer se nahaja stran `meteovista.si` pošlje zahtevek. Njihov strežnik vrne odziv, iz katerega potem izluščimo podatke.

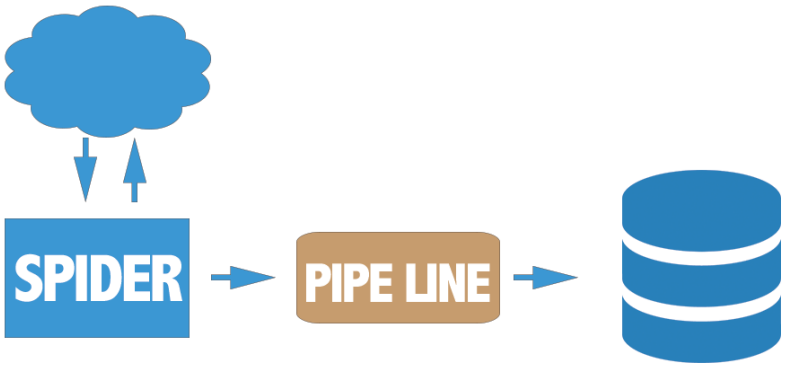
```
class VremeSpider(scrapy.Spider):
    name = "vreme"
    allowed_domains = ["www.meteovista.si"]
```

```
start_urls = [
    "http://www.meteovista.si/Evropa/Slovenija/Piran/4120958",
]
snip_id = 1

def __init__(self, snipid=None, *args, **kwargs):
    super(VremeSpider, self).__init__(*args, **kwargs)
    self.snip_id = snipid

def parse(self, response):
    nizo = response.css(".wo-temperatures strong::text").extract()
    s_item = ScrapypartItem()
    s_item['snip_id'] = self.snip_id
    s_item['data_1'] = nizo[2][:-1]
    s_item['data_1_type'] = "jutranja temperatura - int"
    s_item['data_2'] = nizo[3][:-1]
    s_item['data_2_type'] = "popoldanska temperatura - int"

    yield s_item
```

V zgornji programski kodi si lahko ogledamo preprost primer luščenja podatkov s pomočjo funkcije `css()`. Ta funkcija kot argument prejme niz preko katerega cilja HTML element, iz katerega potem razberemo podatke.

Vse podatke pajek shranjuje v objekt, imenovan `ScrapyItem`, ki v ogrodju Scrapy služi kot nekakršen model za zhranjevanje podatkov. Njegove razrede definiramo v datoteki `items.py`. Po končanem luščenju pajek te objekte pošlje v cevovod, kjer podatke shranjuje v podatkovno bazo. Za shranjevanje teh podatkov sem uporabil knjižnico, imenovano DjangoItem[39], ki objekte `ScrapyItem` preslika v Django modele in jih shrani v podatkovno bazo.

Spletna stran Meteo je statična spletna stran, kjer sta podatka o jutranji in popoldanski temperaturi, upodobljena v HTML, že na strežniku. Podatke lahko luščimo tudi iz dinamičnih spletnih strani, ki svoje podatke v spletno



Slika 5.3: Shema toka podatkov.

stran upodabljajo s pomočjo Javascripta na strani odjemalca. Primer tega so SPA (angl. single page aplication), napisane v programskih ogrodjih, kot je AngularJS [40] ali podobnih. Za luščenje teh strani potrebujemo še dodaten gonilnik, ki nam pomaga pri upodabljanju Javascripta.

```
def __init__(self,  *args, **kwargs):
    self.driver = webdriver.Firefox()
    super(IndiegogoSpider, self).__init__(*args, **kwargs)

def parse(self, response):
    self.driver.get(response.url)
    time.sleep(10)
    e = self.driver.find_elements_by_class_name("i-img")
```

V programskem jeziku Python nam tak gonilnik (angl. driver) ponuja knjižnica Selenium[41]. V zgornji delu programske kode je prikazana uporaba gonilnika. Najprej je potrbeno gonilnik inicializirati, kar neredimo v metodi `__init__`. Pokličemo ga s klicem `webdriver.Firefox()`, ki našemu razredu `IndiegogoSpider` nastavi gonilnik. Ko je gonilnik izbran v metodi `parse()`, pridobimo odgovor. Preden se lahko lotimo luščenja strani, damo program

v spanje. Teh 10 sekund mu da dovolj časa, da gonilnik upodobi in napolni vse značke s podatki preko Javascripa.

5.2.2 Preverjanje pravila

Po luščenju aktualnih podatkov in njihovem shranjevanju v podatkovno bazo je na vrsti preverjanje pravil. Ko podatke preverjamo v majnem intervalu (npr. manj kot eno minuto), se lahko zgodi, da se podatki sploh niso spremenili. Če se podatki na spletni strani niso spremnili, je preverjanje pravila nesmiselno, ker bomo ponovno dobili isto obvestilo. V primeru, ko se podatki spremenijo, pa v pogledu kličem funkcijo `preveri_pravilo(user_snip_id)`. Funkcija preveri nastavljeno pravilo tako, da primerja podatke. Če podatki ustrezajo, pokliče funkcijo `def parse_notification(odrezek)`, ki shrani in vrne novo obvestilo.

5.2.3 Vračanje odziva JSON

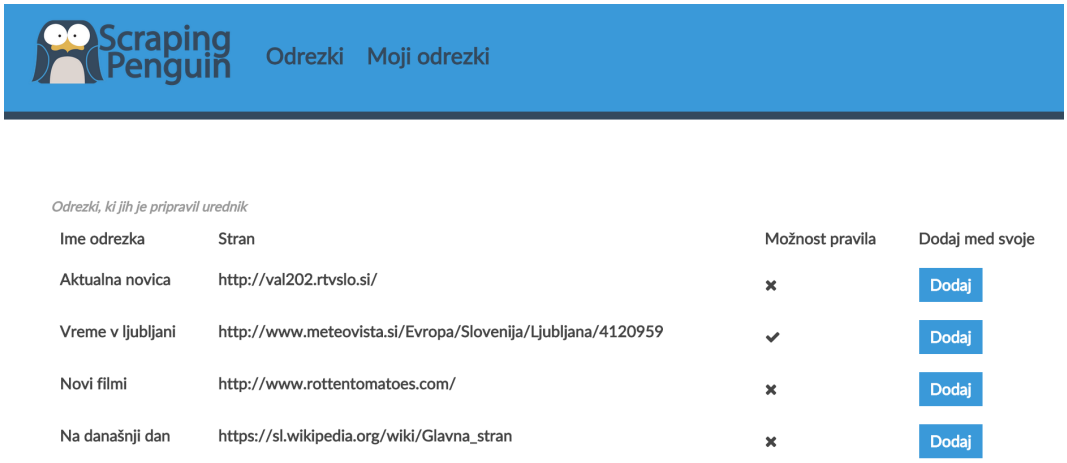
Obvestila, ki jih vrne funkcija `preveri_pravilo(user_snip_id)`, zberemo v seznam in jih preko metode, definirane v `models.py`, vrnemo kot odgovor zahtevku. Spodaj je prikazana oblika modela za obvestila.

```
class Notification(models.Model):
    name = models.CharField(max_length=100)
    desc = models.CharField(max_length=250)
    seen = models.IntegerField(default=0) # 0=neprebrano 1=prebramo

    user_snip_id = models.ForeignKey(
UserHasSnip, db_column='user_snip_id',
related_name='user_snip_id', null=True)

    def json(self):
        return {
            'name': self.name,
```

```
        'desc': self.desc,
    }
```

5.3 Podstran odrezki

Na te podstrani se uporabniku prikaže tabela odrezkov, izbranih s strani urednika spletne aplikacije. Zraven imena in strani, iz katere se prikazuje odrezek, tukaj vidimo tudi, če ima odrezek možnost definiranja pravila. Pravilo lahko določimo vsem odrezkom, ki imajo prej definirane pajke. Uporabnik si lahko s klikom na gumb `Dodaj` odrezek doda med svoje.



Slika 5.4: Pregled odrezkov, ki jih je izbral urednik.

5.4 Moji odrezki

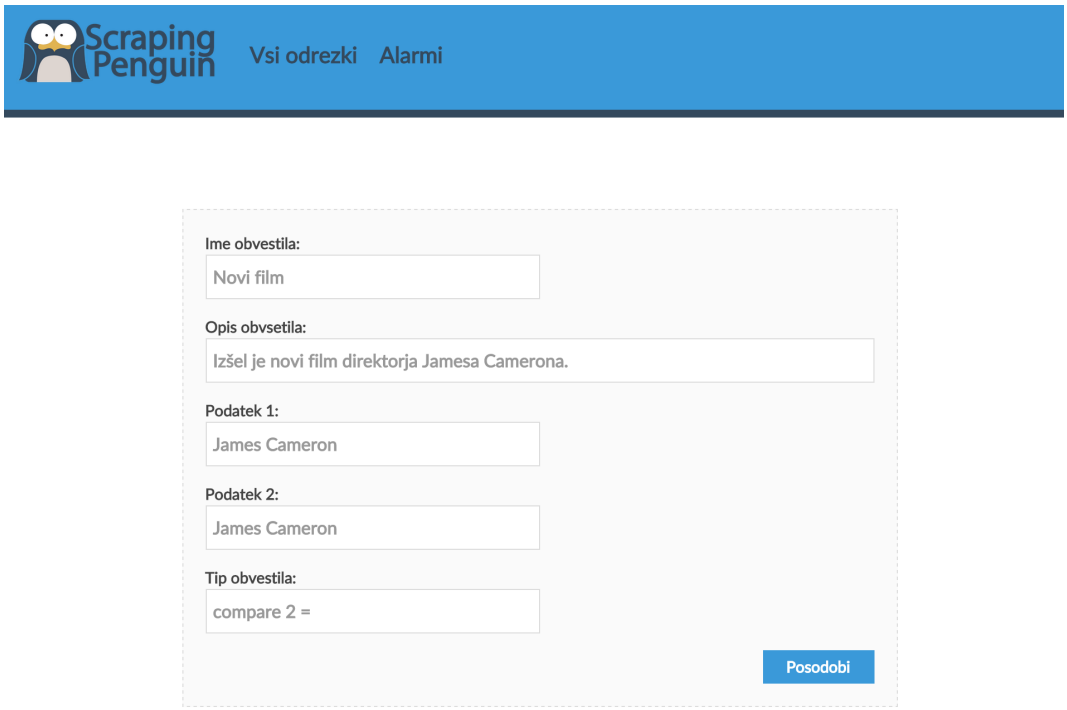
Podstran `Moji odrezki` služi kontroliranju prikaza uporabniških odrezkov preko dveh tabel. Prva tabela nam prikazuje seznam odrezkov, ki se prikažejo na domači strani. S klikom na gumb `Prilagodi` spletna stran uporabnika preusmeri na podstran prilagodi, kjer ima možnost urejanja prikaza izbrnega

odrezka. Odrezkom, ki imajo definirane pajke lahko preko gumba `Nastavi pravilo` nastavimo pravilo, preko katerega nam potem sistem streže obvestila.

Slika 5.5: Podstran moji odrezki.

5.4.1 Uporabniško pravilo

Uporabnik si lahko preko tega obrazca, prikazanega na Sliki 5.6, nastavlja pravilo. Obvezno mora vnesti ime, opis in vrsto obvestila. Ko vnese te tri podatke, se omogoči gumb `Nastavi`, in ob kliku nanj se nastavi pravilo. V vnosnem polju `Tip obvestila` si uporabnik izbere vrsto pravila. Kot vhod sprejme niz, sestavljen iz treh besed, ločenih s presledkom. Prva predstavlja vrsto in podpira besedi `compare`, ki pomeni primerjaj in `new`, ki pomeni preveri, ali se je na strani spremenil podatek. Z drugo besedo določimo če ciljamo prvi podatek, drugi ali kar oba. Zadnja beseda predstavlja operator enako, manjše ali večje.



Slika 5.6: Nastavljanje uporabniškega pravila.

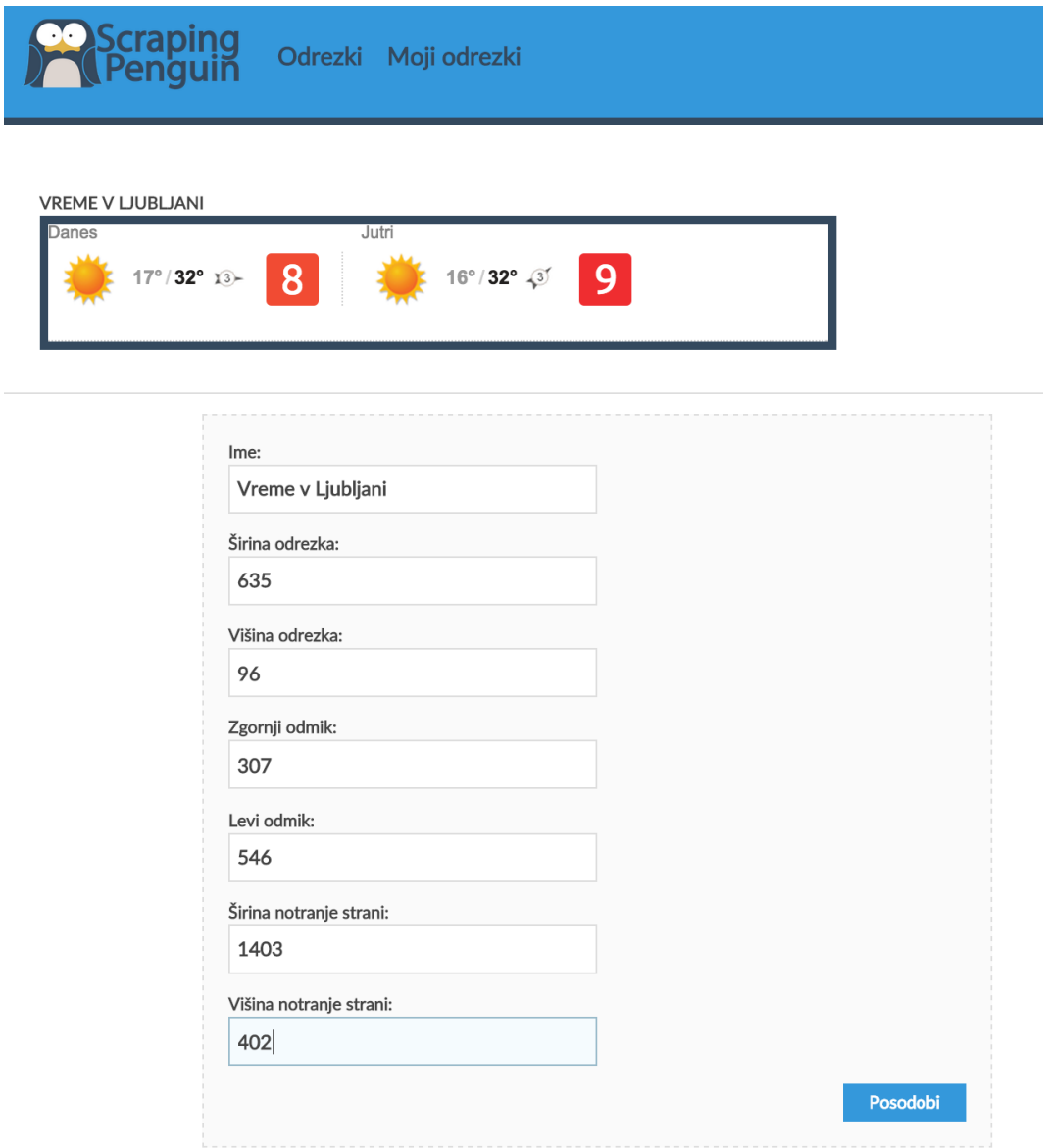
Preko vnosnega polja `Podatek 1` in `Podatek 2` uporabnik vnese niza, s katerima kasneje sistem primerja podatke, pridobljene iz drugih strani. Primerja jih preko operatorja izbranega v vnosnem polju.

5.4.2 Urejanje odrezka

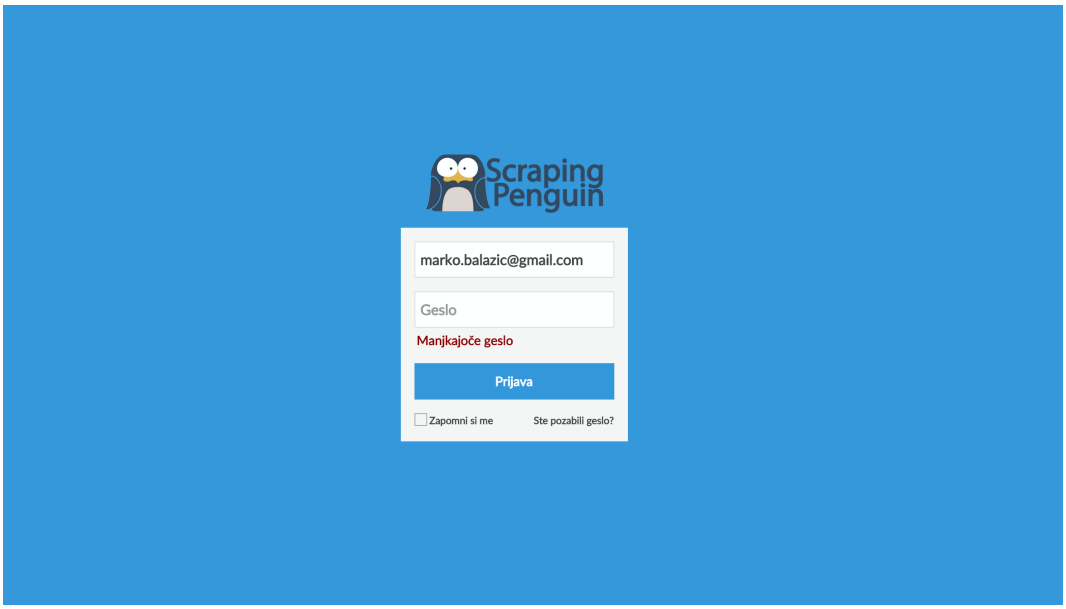
Če si je uporabnik odrezek dodal preko razširitve, torej odrezek ni bil izbran iz uredniških, potem ima uporabnik možnost urejanja tega odrezka. To možnost sem vključil predvsem zato, ker spletne strani velikokrat spreminjajo svojo strukturo. Če se spremeni spletna stran, se spremeni tudi prikaz odrezka. Vsako polje ima nase pritrjenega tudi poslušalca, ki ob kliku izven polja sinhronizira podatek s predogledom odrezka. Ta sinhornizacija pripomore k bolj natančnemu urejanju odrezka in k boljši uporabniški izkušnji.

5.5 Prijavni zaslon

Aplikacijo sem zasnoval tako, da podpira različne uporabniške vloge. Taka zasnova bi pri morebitnem nadaljnem razvoju pripomogla k lažjem nadzoru vsebine. Na prijavni strani se torej lahko prijavijo tako urednik kot navadni uporabniki. Pri prijavi morajo vnesti `Email` in `Geslo`. Za preverjanje pravilnosti sem spisal validacijo obrazca. Validacija preverja pravilnost prvega in drugega polja. Pri prvem preverja, če je uporabnik vnesel pravilen elektronski naslov, torej če ima v nizu znak `@` in za njim domeno. Elektronski naslov primerja tudi s podatkovno bazo. Ta primerjava vrne napako, če v bazi ne najde vnosa. Pri drugem vnosnem polju validacija preverja dolžino gesla in če se geslo sklaja z elektronskim naslovom.



Slika 5.7: Urejanje odrezka vreme.



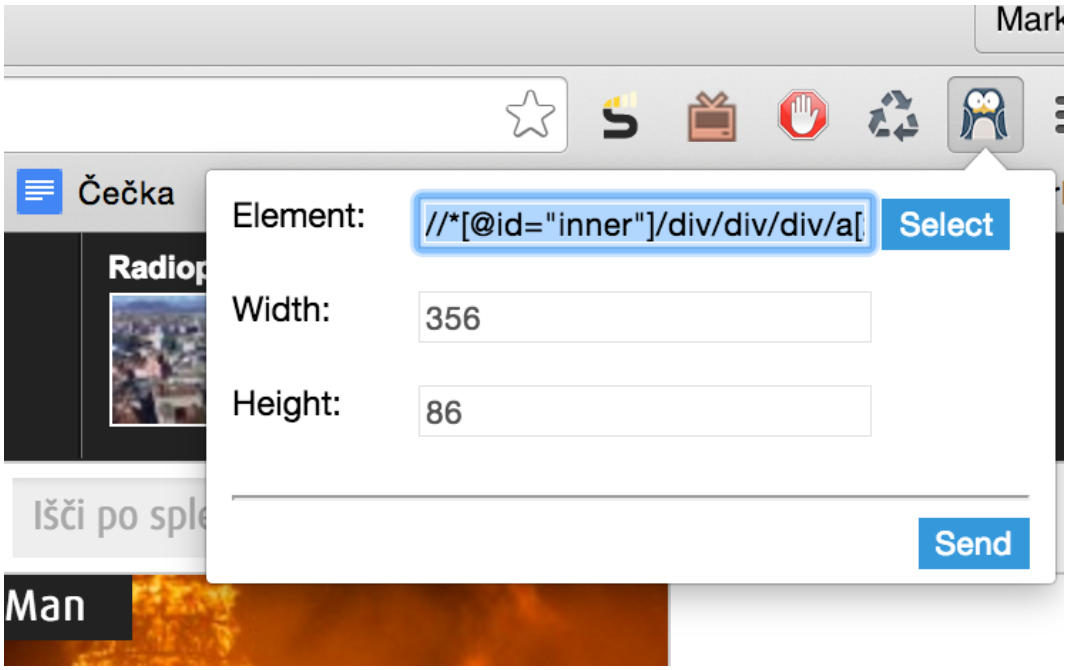
Slika 5.8: Prikaz validacije na prijavnem zaslonu.

5.6 Google Chrome razširitev

5.6.1 Arhitekura

Poznamo dve vrsti razširitev `persistent: true` in `false`. Persistent nastavljen, na `true`, pomeni, da se glavna skripta `backgound.js` izvaja neprestano v ozadju brskalnika. Če pa persistent nastavimo na `false`, se skripa izvede le ob klicu in se po koncu vrne v mirovanje. Če le gre je boljše da izklopimo možnost persistent, saj tako omogočimo boljše delovanje brskalnika [42]. Razširitev je sestavljena iz treh glavnih skript zapisanih v Javascriptu, ki skupaj skrbijo za delovanje:

- `backgournd.js` - Je glavna skripta, ki se izvaja v ozadju. Preko nje se prožijo tudi oddaljeni klici na strežnik. V mojem primeru skripta operira in shranjuje opisne vrednosti odrezka, ki jih prejme iz skripte `content.js`.
- `popup.js` - Je skripta, ki kontrolira dogodke v prikaznem oknu in polni



Slika 5.9: Prikazno okno, del Google razširitve.

vhodna polja z opisnimi podatki o odrezku.

- `content.js` - Ta skripta se vključi v DOM drevo izbrane strani, kjer zbira podatke o odrezku in jih pošilja naprej.

Pri razširitvah imamo več možnosti vključitve uporabniškega vmestnika. Sam sem izbral možnost prikaznega okna, ki je prikazana na sliki 5.9. Preko vmesnika s klikom na `Select` uporabnik začne z izbiro elementa, ki ga želi uporabiti kot odrezek. Ko je element izbran, se uporabniku v prikaznem oknu pokažejo trije podatki in omogoči se možnost `Send`, ki podatke pošlje v uporabnikov profil.

5.6.2 Content.js

Glavna naloga razširiteve je zajem odrezka. Temu je namenjena ta skripta. Ko uporabnik začne z izbiro elementa, skripta zažene funkcijo, ki ob pre-

mikanju miške izrisuje zelen kvadrat v velikost elementa, ki se nahaja pod miško. Ta del postopka je prikazan na Sliki 5.10.

Slika 5.10: Prikaz zajema elementa z uporabo razširitve.

Ob kliku na izbran element se podatki shranijo v polnilnik brskalnika, čez spletno stran pa se nam izpiše obvestilo z napisom "Element izbran". Večji del skripte `content.js` je zapisan s pomojčjo knjižnice jQuery.

```
$("h1,h2,h3,div,span,a,img,section,header,footer").hover(function(){
        $("#hover-overlay")
            .css("left", $(this).offset().left)
            .css("top", $(this).offset().top)
            .width($(this).outerWidth())
            .height($(this).outerHeight())
            .show();
    },function(){
        $("#hover-overlay").hide();
        $(".dialogs").remove()
    });
```

Zgornja programska koda nam ob premikanju miške izrisuje zelen kvadrat.

Poglavje 6

Kritičen pogled in možne izboljšave

Verjamem, da je možno vsak koncept izboljšati in za stopnjevanje so potrebne kritike. To poglavje je namenjeno ravno temu. Skozi razvoj celotnega sistema sem velikokrat prišel do točke, kjer zaradi take ali drugačne omejitve nisem mogel naprej. Kot primer ene takih omejitev bi lahko izpostavil XSS, ki me je prisilil k uporabi spletnih pajkov.

Eno večjih pomanjkljivosti sistema vidim v tem, da se obvestila preverjajo samo takrat, ko je uporabnik prisoten na spletni strani, ali pa jo ima odprto v zavihku. Slednje bi se dalo izboljšati z uporabo podprocesov, ki bi ob določenem intervalu klicali servis za luščenje in uporabnika v realnem času obveščali o spremembah. Tako bi zmanjšali odzivni čas med dejanskim novim podatkom na spletni strani, ki jo luščimo, in med obvestilom na spletni aplikaciji. Če predpostavimo to nadgradnjo, bi lahko uporabniki takoj dobili sporočila preko elektronske pošte ali tekstovnega sporočila.

Sistem bi se dalo izboljšati tudi z uvedbo SPA (angl. single page application), ki bi vse spremembe vključno z obvestili pošiljal v ozadju, brez da bi osveževal celotno spletno stran. Vse spletne strani napisane, v stilu SPA, imajo še eno pomembno prednost, ki omogoča ponoven izris delov grafičnega vmesnika. To prednost bi lahko uporabili pri urejanju prikaza enega odrezka,

npr. uporabnik bi preko vnosnega polja vpisal širino odrezka in ta bi se brez dodatnega poslušalca spremenila.

Poglavje 7

Zaključek

Pri pisanju diplomskega dela sem pridobil veliko znanja o programiranju spletnih luščilcev podatkov ali krajše pajkov. Proces raziskovanja možnih rešitev me je pripeljal do končne arhitekure brskalnik-servis-strežnik, ki je omogočala izvedbo zastavljenih ciljev. Naučil sem se tudi načine povezave med različnimi deli celotnega sistema. Pri končni rešitvi sem med seboj povezal razšitev za brskalnik Chrome, strežnik, napisan v ogrodju Django, ki je služil kot glavna komunikacija med vsemi deli, in servis, napisan v ogrodju Scrapy.

Rahlo spremenjen projekt sem prijavil tudi na natečaj idej, kjer je zmagal. Projekt se sedaj še naprej razvija v sklopu podjetja, ki je razpisalo natečaj. Oktobra bom projekt tudi predstavil na slovenski konferenci "Science & bussines".

Literatura

[1] How browsers sees a web page. *Javascript & jQuery* - Jon Ducket , 2013.

[2] Semantic web. [Online]. Dosegljivo: http://www.cs.umd.edu/ golbeck/LBSC690/SemanticWeb.html [Dostopano 28. 8. 2015].

[3] Web scraping. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Web_scraping [Dostopano 19. 8. 2015].

[4] Media tool kit. [Online]. Dosegljivo: https://www.mediatoolkit.com/_development [Dostopano 1. 9. 2015].

[5] Skyscenner. [Online]. Dosegljivo: https://www.skyscanner.net [Dostopano 1. 9. 2015].

[6] Report: Bot traffic is up to 61.5% of all website traffic. [Online]. Dosegljivo: https://www.incapsula.com/blog/bot-traffic-report-2013.html [Dostopano 21. 8. 2015].

[7] Meta tags that Google understands. [Online]. Dosegljivo: https://support.google.com/webmasters/answer/79812?hl=en [Dostopano 28. 8. 2015].

[8] Robot.txt . [Online]. Dosegljivo: https://www.feedthebot.com/robottxt.html [Dostopano 28. 8. 2015].

[9] Python. [Online]. Dosegljivo: https://www.python.org/doc/essays/blurb/ [Dostopano 19. 8. 2015].

[10] Python displacing R as the programming language for data science. [Online]. Dosegljivo: http://readwrite.com/2013/11/25/python-displacing-r-as-the-programming-language-for-data-science [Dostopano 19. 8. 2015].

[11] PyPI - Python. [Online]. Dosegljivo: https://pypi.python.org/ [Dostopano 19. 8. 2015].

[12] Pyhon vs C speed. [Online]. Dosegljivo: http://theunixgeek.blogspot.com/2008/09/c-vs-python-speed.html [Dostopano 19. 8. 2015].

[13] HTML 5. [Online]. Dosegljivo: http://www.w3.org/TR/html5/ [Dostopano 28. 8. 2015].

[14] Standard ECMA-262 [Online]. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf [Dostopano 5. 9. 2015].

[15] Why use jQuery? *Javascript & jQuery* - Jon Ducket , 2013.

[16] Standard ECMA-262 [Online]. http://blog.jquery.com/2014/01/13/the-state-of-jquery-2014/ [Dostopano 5. 9. 2015].

[17] Xpath documentation - Section Introdaction [Online]. http://www.w3.org/TR/xpath/#section-Introduction [Dostopano 5. 9. 2015].

[18] Python Package Index [Online]. https://pypi.python.org/pypi [Dostopano 5. 9. 2015].

[19] Django - glavna stran. [Online]. Dosegljivo: https://www.djangoproject.com/ [Dostopano 28. 8. 2015].

[20] *Django Documentation - Release 1.8.1* - Django software foundation, 2015.

[21] Regular expresions. [Online]. Dosegljivo: http://www.tutorialspoint.com/unix/unix-regular-expressions.htm [Dostopano 19. 8. 2015].

[22] Kratka zgodovina Gita. [Online]. Dosegljivo: http://git-scm.com/book/sl/v2/Pri%C4%8Detek-Kratka-zgodovina-Git-a [Dostopano 20. 8. 2015].

[23] Git basics. [Online]. Dosegljivo: http://git-scm.com/book/en/v2/Getting-Started-Git-Basics [Dostopano 27. 8. 2015].

[24] Google extensions wiki. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Google_Chrome_extension [Dostopano 19. 8. 2015].

[25] Extensions Overview. [Online]. Dosegljivo: https://developer.chrome.com/extensions/overview_software_development [Dostopano 31. 8. 2015].

[26] Google Web Store. [Online]. Dosegljivo: https://chrome.google.com/webstore/category/apps?utm_source=chrome-ntp-icon. [Dostopano 19. 8. 2015].

[27] Brackets. [Online]. Dosegljivo: https://github.com/adobe/brackets [Dostopano 19. 8. 2015].

[28] Google going its own way forking webkit rendering engine. [Online]. Dosegljivo: http://arstechnica.com/information-technology/2013/04/google-going-its-own-way-forking-webkit-rendering-engine/ [Dostopano 19. 8. 2015].

[29] Browser share. [Online]. Dosegljivo: http://www.w3schools.com/browsers/browsers_stats.asp [Dostopano 19. 8. 2015].

[30] DOM. [Online]. Dosegljivo: http://www.w3.org/DOM/ [Dostopano 30. 8. 2015].

[31] Source tree application [Online]. https://www.sourcetreeapp.com/ [Dostopano 5. 9. 2015].

[32] Google dokumenti. [Online]. Dosegljivo: https://www.google.com/docs/about/ . [Dostopano 19. 8. 2015].

[33] Operacijski sistem OS X. [Online]. Dosegljivo: http://www.apple.com/osx/ [Dostopano 30. 8. 2015].

[34] Web Clip widget. [Online]. Dosegljivo: https://support.apple.com/kb/PH14092?locale=sl_SI. [Dostopano 19. 8. 2015].

[35] User story. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/User_story [Dostopano 23. 8. 2015].

[36] Cross site scripting. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Cross-site_scripting [Dostopano 19. 8. 2015].

[37] MySQLWorkbench. [Online]. Dosegljivo: https://www.mysql.com/products/workbench/ [Dostopano 19. 8. 2015].

[38] Agile software development. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Agile_software_development [Dostopano 30. 8. 2015].

[39] Django item. [Online]. Dosegljivo: https://pypi.python.org/pypi/scrapy-djangoitem [Dostopano 29. 8. 2015].

[40] AngularJS. [Online]. Dosegljivo:
https://angularjs.org/ [Dostopano 28. 8. 2015].

[41] Selenium. [Online]. Dosegljivo:
http://www.seleniumhq.org/ [Dostopano 30. 8. 2015].

[42] Event Pages. [Online]. Dosegljivo:
https://developer.chrome.com/extensions/event_pagesoverview_software_development
[Dostopano 31. 8. 2015].